Package: brassica (via r-universe)

October 31, 2024

Type Package

Title 1970s BASIC Interpreter

Version 1.0.2

Date 2022-10-24

Author Mike Lee

Maintainer Mike Lee <random.deviate@gmail.com>

Description Executes BASIC programs from the 1970s, for historical and educational purposes. This enables famous examples of early machine learning, artificial intelligence, natural language processing, cellular automata, and so on, to be run in their original form.

License GPL-3

Depends R (>= 3.3.0)

NeedsCompilation no

Repository CRAN

Date/Publication 2022-10-24 06:15:06 UTC

Contents

brassica-package *BASIC Interpreter*

Description

Runs 1970s BASIC programs.

Details

Interprets and executes a subset of early (1975) Altair/Microsoft BASIC, plus some generalisations and extensions. Enables various programs of historical interest to be run in their primal form. A selection of public-domain examples are included.

Author(s)

Mike Lee

See Also

RUN

Examples

```
require(brassica)

## Not run:
# Load and run the 'Wumpus' program.
RUN("WUMPUS")

## End(Not run)
```

LIST *List BASIC Programs*

Description

Lists whichever BASIC program is loaded.

Usage

```
LIST()
```

Value

Returns a character vector, containing the complete listing of whichever BASIC program is in memory at the time.

See Also

NotRun

Examples

```
# List the current BASIC program.
LIST()
```

LowerCode *Convert to Lower Case*

Description

Converts a BASIC program to lower case, except for any string literals.

Usage

```
LowerCode(p)
```

Arguments

p	A BASIC program listing, as a character vector.

Value

Returns the converted listing, as a character vector.

See Also

UpperCode

Examples

```
# Convert a program to lower case.
LowerCode(c('10 LET X=X+1', '20 PRINT "Hello, World!"'))
```

NotRun *Check BASIC Programs*

Description

Returns a data frame of any BASIC program lines that contain one or more unexecuted statements. Used with repeated calls of `RUN()`, this provides a test of code syntax and accessibility.

Usage

```
NotRun(pretty = TRUE)
```

Arguments

`pretty`	A single logical value. If `TRUE` (the default), formatted output is printed before the data frame is returned invisibly. If `FALSE`, the data frame is returned normally (visibly), without any other output being printed.

Value

Returns a data frame of any lines, from the currently-loaded BASIC program, containing one or more unexecuted statements. Lines are listed with both their BASIC line numbers and the corresponding file line numbers of the source script. A persistent appearance here may, but does not necessarily, signify a problem with the line.

Note

BASIC `ON` and `IF-THEN-ELSE` statements contain multiple alternative clauses. Each such statement, as a whole, is marked as having been executed once any one of its clauses has been run without error. Hence, the absence of such a statement from the `NotRun()` frame does not guarantee that all of its clauses are error-free.

See Also

`RUN`

Examples

```
# Peruse which lines have not yet been run.
NotRun()
```

RUN *Run BASIC Programs*

Description

Loads and runs BASIC programs.

Usage

```
RUN(program = NULL, ttx = 0, tty = 0, up = FALSE)
```

Arguments

program	A character string, containing the name or file path of a BASIC program. Omitting this re-runs the currently-loaded program.
ttx	Aesthetic teletype option. A delay, of not more than 200 milliseconds, to be applied after printing each character. In practice, the actual delay will be this time plus however long it takes to flush the console, plus interpreter overheads. Depending on the model, a real teletype could manage around ten characters per second. Making this negative imposes no delay, but flushes the console after each completed line (not after each character).
tty	Aesthetic teletype option. A delay, of not more than 2000 milliseconds, to be applied after each line of printing. Making this negative imposes no delay, but still flushes the console after each completed line.
up	Aesthetic teletype option. Making this TRUE forces all BASIC output to upper case.

Details

List of bundled example programs (from the references):

Animal	by Arthur Luehrmann	(decision trees)
Camel	by the Heath Users Group	(random walk)
Chase	by Mac Oglesby	(robots)
Eliza	by Jeff Shrager	(language processing)
Even Wins	by Eric Peters	(machine learning)
Flip	by John S. James	(machine learning)
Four in a Row	by James L. Murphy	(artificial intelligence)
Guess-It	by Gerard Kierman	(bluffing)
Hammurabi	by David H. Ahl	(resource management)
Hexapawn	by R. A. Kaapke	(machine learning)
Inkblot	by Scott Costello	(projection)
Life	by Clark Baker	(cellular automata)
Maze	by Richard Schaal	(depth-first search)
Not One	by Robert Puopolo	(expectation values)
Sea Battle	by Vincent Erickson	(monsters)

Super Star Trek	by Richard Leedon	(vector space)
Wumpus	by Gregory Yob	(graph topology)

All of these are in the public domain.

Value

Returns an invisible `NULL`, after printing to standard output.

Note

Many BASIC programs ask for user input after printing a custom prompt. When `ttx` is greater than zero, some R sessions may incorrectly position the cursor at the beginning of the line, or on the next. Neither occurrence is fatal. Known to be affected: Windows R terminal ('`R`' and '`Rterm`', but the '`Rgui`' console is fine), Mac R console (but the Bash terminal is fine), RStudio. Additionally, double buffering, or the like, may interfere with the effect at higher speeds (Mac R console, again).

References

1. David H. Ahl & Steve North, *BASIC Computer Games* (1978)
2. David H. Ahl & Steve North, *More BASIC Computer Games* (1979)

See Also

`LIST`

Examples

```
## Not run:
# Load and run 'Wumpus'.
RUN("wumpus")

# Load and run 'Chase', flushing the
# console after every line of output.
RUN("chase", -1)

# Load and run 'Camel', with full
# retro-style teletypewriter effects.
RUN("camel", 20, 300, TRUE)

# Load and run your program.
RUN("myprogram.bas")

# Re-run the last program
# (without re-loading it).
RUN()

## End(Not run)
```

UpperCode *Convert to Upper Case*

Description

Converts a BASIC program to upper case, except for any string literals.

Usage

```
UpperCode(p)
```

Arguments

p	A BASIC program listing, as a character vector.

Value

Returns the converted listing, as a character vector.

See Also

LowerCode

Examples

```
# Convert a program to upper case.
UpperCode(c('10 let x=x+1', '20 print "Hello, World!"'))
```

Index